Intro to Shiny

Katie Wilson

University of Washington

10/19/2017

What is Shiny?

Shiny is an R package that lets you build interactive web apps right from R! It allows users to change graphic inputs dynamically.

First, make sure to install shiny:

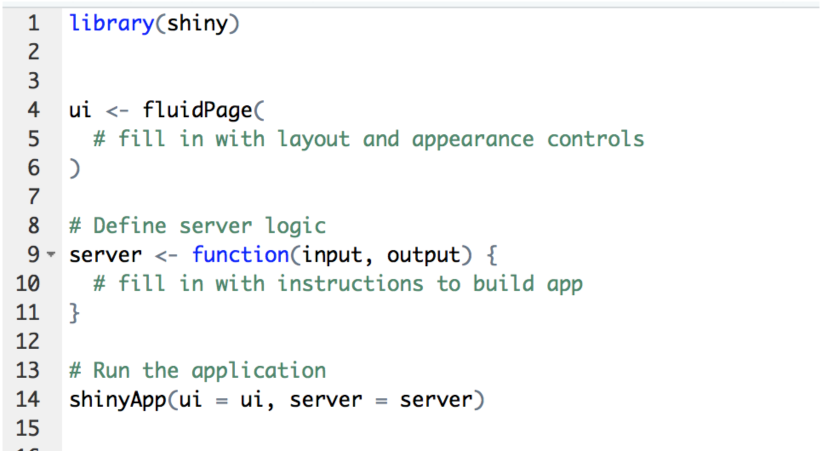
```r
install.packages("shiny")
```

Shiny apps are contained in a single script called `app.R`, which has 3 components:

- a user interface object (`ui`)
- a server function (`server`)
- a call to the `shinyApp` function

Shiny app script skeleton

This would all be in `app.R`

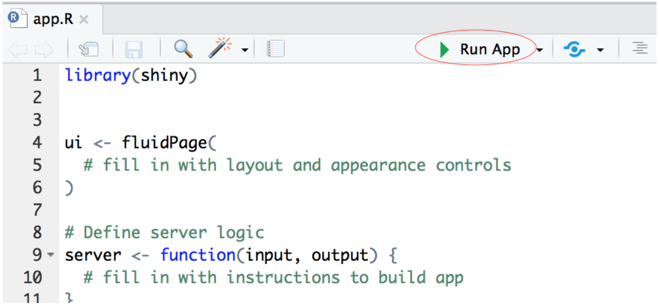


```r
library(shiny)


ui <- fluidPage(
  # fill in with layout and appearance controls
)

# Define server logic
server <- function(input, output) {
  # fill in with instructions to build app
}

# Run the application
shinyApp(ui = ui, server = server)

```

Running the app

There are 2 (recommended) options.

- If using RStudio



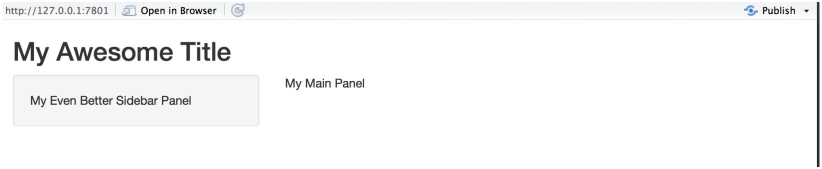
```
library(shiny)


ui <- fluidPage(
  # fill in with layout and appearance controls
)

# Define server logic
server <- function(input, output) {
  # fill in with instructions to build app
}
```

- Or save the file app.R in a folder (e.g. “My-Shiny-App”) and run:

```
library(shiny)
runApp("My-Shiny-App")
```

The UI: introduction

- Use the function `fluidPage` to create the display
- To get this:



My Awesome Title

My Even Better Sidebar Panel

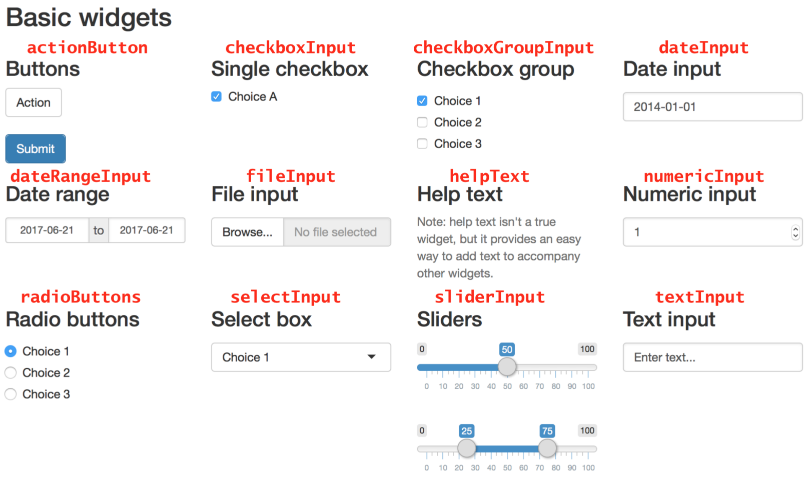
My Main Panel

- Do this:

```r
ui <- fluidPage(
  titlePanel("My Awesome Title"),

  sidebarLayout(
    sidebarPanel("My Even Better Sidebar Panel"),
    mainPanel("My Main Panel")
  )
)
```

The UI: widgets

- You can use widgets to collect a (or many) value(s) from the user

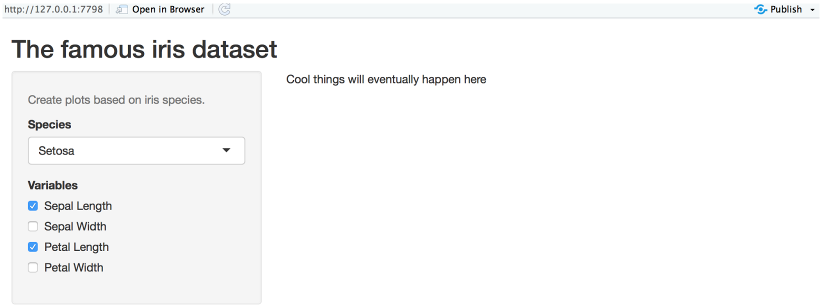


Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

The UI: iris example

```r
ui <- fluidPage(
  titlePanel("The famous iris dataset"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create plots based on iris species."),
      selectInput("type",
                  label = "Species",
                  choices = list("Setosa", "Versicolor", "Virginica"),
                  selected = "Setosa"),
      checkboxGroupInput("vars",
                         label = "Variables",
                         choices = list("Sepal Length", "Sepal Width",
                                        "Petal Length", "Petal Width"),
                         selected = c("Sepal Length", "Petal Length"))
    ),
    mainPanel("Cool things will eventually happen here")
  )
)
```

The UI: iris example



The famous iris dataset

Create plots based on iris species.

Species

Setosa

Variables

- ✓ Sepal Length
- Sepal Width
- ✓ Petal Length
- Petal Width

Cool things will eventually happen here

The UI: adding objects

- We can add R objects to the UI:
    - Plot, table, text
    - among others. . .
- Can be placed inside the sidebarPanel or mainPanel
- Returning to the iris example, suppose we want to include some text and a plot that depends on values from the UI:

```r
ui <- fluidPage(
  titlePanel("The famous iris dataset"),

  sidebarLayout(
    ...
    mainPanel(
      textOutput("selected_species"),
      plotOutput("irisplot")
  )
)
```

The Server: introduction

- Now that we told Shiny where to display our object, we need to tell Shiny how to build the object
- To do this we use the `server` function
- It will take `input`, which is a list-like object storing current values of all widgets in the app. Recall the names you used in the `ui` (`type` and `vars` in the iris case)
- It will produce `output`, which should contain the output of one of Shiny's `render*` functions
  - `renderPlot`, `renderTable`, `renderText`
  - among others

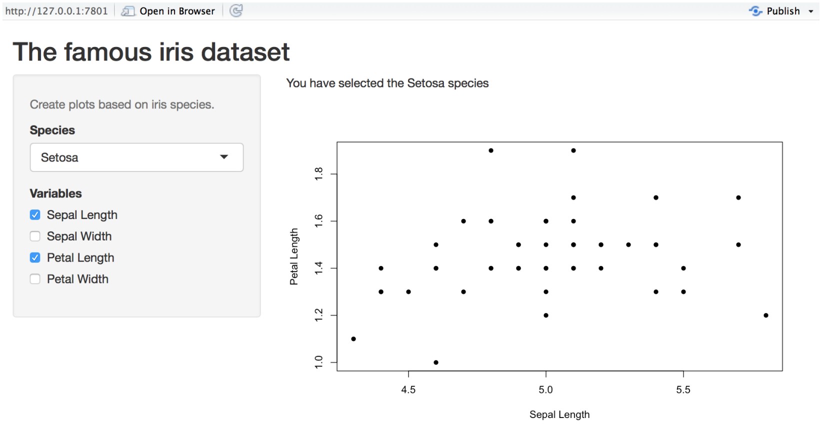
The Server: iris example

- Recall: `type` is the name of the iris species and `vars` is a 2-dim vector containing the variables
- Also recall what the iris dataset looks like:

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

```
server <- function(input, output) {
  output$selected_species <- renderText({
    paste("You have selected the", input$type, "species")
  })

  output$irisplot <- renderPlot({
    data.to.plot <- iris[iris$Species == tolower(input$type), ]
    x <- data.to.plot[, gsub(" ", ".", input$vars[1])]
    y <- data.to.plot[, gsub(" ", ".", input$vars[2])]
    plot(x, y, xlab=input$vars[1], ylab=input$vars[2], pch=16)
  })
}
```

The Server: iris example



http://127.0.0.1:7801 Open in Browser Publish

The famous iris dataset

Create plots based on iris species.

Species

Setosa

Variables

- [✓] Sepal Length
- [ ] Sepal Width
- [✓] Petal Length
- [ ] Petal Width

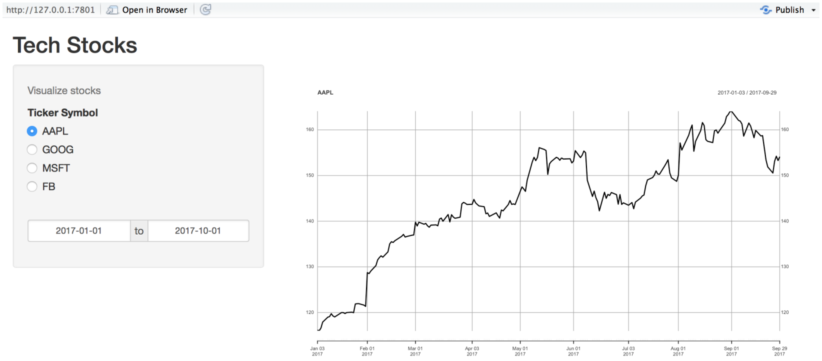
You have selected the Setosa species

Conclusion

- Shiny is an awesome way to visualize information
- You can make way more complicated Shiny apps
- Check out https://shiny.rstudio.com/ for videos and written tutorials on all the things you can do with Shiny!

Stocks

As another example, let's suppose we want to examine stock prices of some tech companies. Pretend that this is what we are after:



Tech Stocks

Visualize stocks

Ticker Symbol

[✓ AAPL]
GOOG
MSFT
FB

2017-01-01 to 2017-10-01

You will need to use the package `quantmod` to obtain stock prices from Yahoo

Stocks: skeleton

```r
library(shiny)
library(quantmod)

ui <- fluidPage(
  titlePanel("Tech Stocks"),

  sidebarLayout(
    sidebarPanel(
      helpText( ... ),
      radioButtons( ... ),
      dateRangeInput( ... )
    ),
    mainPanel( ... )
  )
)

# Define server logic for making the plot
server <- function(input, output) {
  ...
}

# Run the application
shinyApp(ui = ui, server = server)
```

Stocks: UI

```r
ui <- fluidPage(
  titlePanel("Tech Stocks"),

  sidebarLayout(
    sidebarPanel(
      helpText("Visualize stocks"),
      radioButtons("stock",
                   label = "Ticker Symbol",
                   choices = list("AAPL", "GOOG", "MSFT", "FB"),
                   selected = "AAPL"),
      dateRangeInput("dates",label="",
                     start="2017-01-01", end="2017-10-01",
                     min="2017-01-01", max="2017-10-01")
    ),
    mainPanel(
      plotOutput("stockplot")
    )
  )
)
```

Stocks: server

```r
server <- function(input, output) {
  output$stockplot <- renderPlot({
    start <- as.Date(input$dates[1])
    end <- as.Date(input$dates[2])

    getSymbols(input$stock, src = "yahoo", from = start,
               to = end,
               warnings=FALSE)
    plot(eval(parse(text=input$stock))[, paste0(input$stock,".Close")],
         main = input$stock)
  })
}
```